

# RIVIC QSSP PLATFORM

## Comprehensive Technical White Paper

Quantum-Safe Cryptographic Build System with Cloud-Native Foundation

**Version:** 1.0

**Date:** December 26, 2025

**Status:** Production Architecture Specification

**Scope:** Full-Stack Enterprise Deployment

---

## EXECUTIVE SUMMARY

The **Rivic Quantum-Safe Secure Platform (QSSP)** is an enterprise-grade, cloud-native cryptographic infrastructure designed for post-quantum cryptography (PQC) migration, eIDAS 2.0 compliance, and quantum-safe digital identity. This platform enables financial institutions, government agencies, and critical infrastructure operators to transition from classical RSA/ECC cryptography to NIST-approved quantum-resistant algorithms while maintaining backward compatibility and regulatory compliance.

## Key Capabilities

- **PQC Migration Framework:** Hybrid RSA + ML-KEM (NIST FIPS 203) with zero-downtime transition
- **Quantum Key Exchange:** BB84 protocol implementation with simulated quantum randomness
- **Cloud-Native Architecture:** Kubernetes-native, containerized quantum computing stack
- **eIDAS 2.0 Compliance:** Advanced Electronic Signature (AdES) with post-quantum algorithms
- **IBM Attestation Integration:** Secure enclave operations with hardware-backed cryptographic verification
- **CBOM Tracking:** Complete cryptographic bill of materials for supply chain visibility
- **Zero-Trust Security Model:** Attestation-driven access control and cryptographic authorization

## Target Markets

1. **Financial Services** - Banking, insurance, payment systems
2. **Government & Defense** - eIDAS digital signatures, classified document protection

3. **Healthcare** - HIPAA-compliant cryptographic infrastructure
4. **Supply Chain** - Blockchain, IoT device authentication
5. **Critical Infrastructure** - Energy grid, water systems, telecommunications

## Project Timeline

Phase	Duration	Deliverables	Status
<b>Phase 1: Core Platform</b>	Q1-Q2 2026	Python/C++ quantum bridge, Kubernetes operators, hybrid crypto	In Progress
<b>Phase 2: IBM Integration</b>	Q3 2026	Attestation API, TEE integration, compliance certification	Planned
<b>Phase 3: eIDAS Compliance</b>	Q4 2026	Digital signature framework, legal validation	Planned
<b>Phase 4: Enterprise Release</b>	Q1 2027	Production support, consulting services	Planned

## SECTION 1: PROJECT SCOPE & GOVERNANCE

### 1.1 Scope Statement

The Rivic QSSP Platform encompasses:

#### In Scope

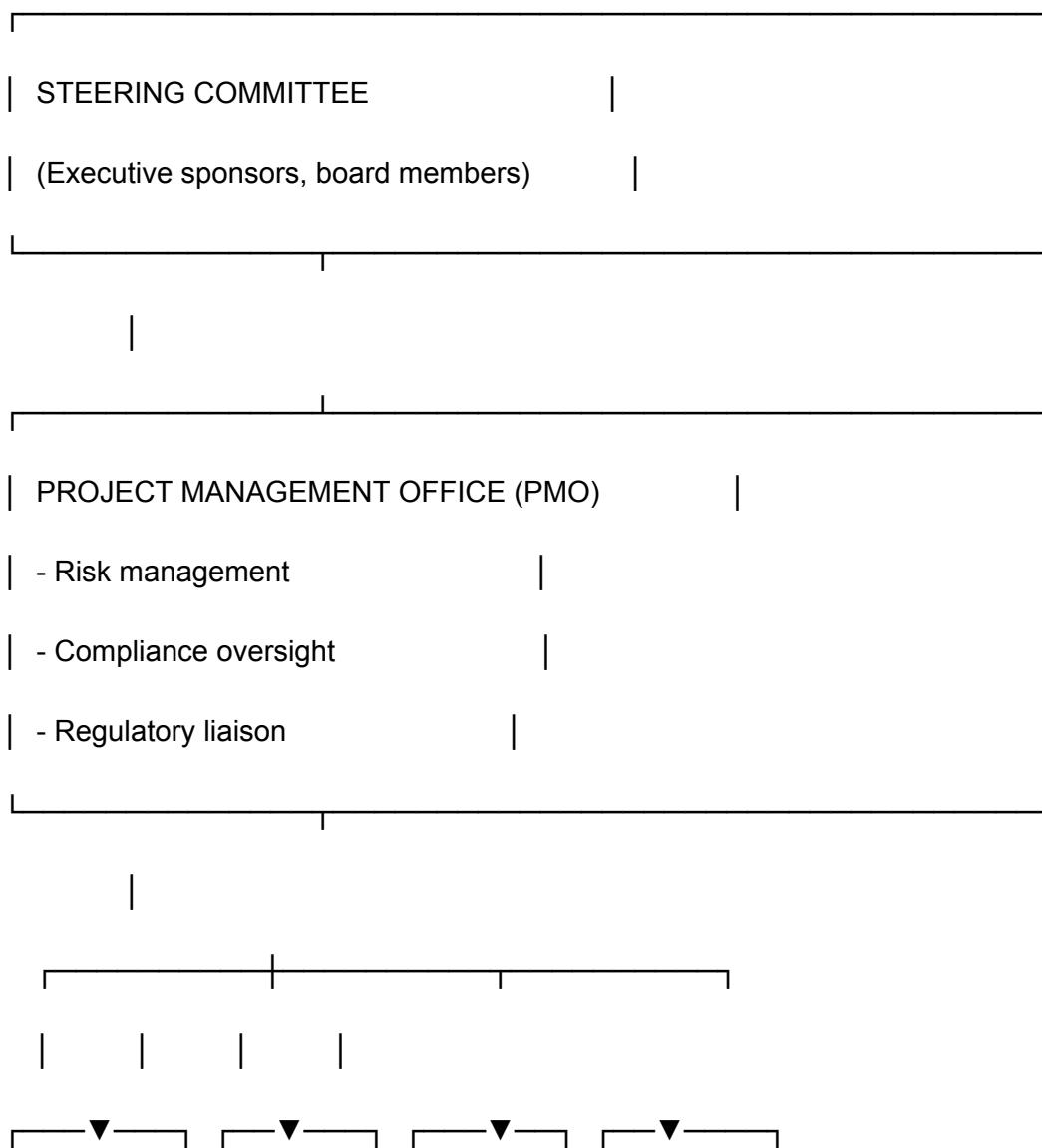
- Hybrid RSA-2048 + ML-KEM-1024 cryptographic engine
- Python/C++ quantum computing interface (Qiskit + liboqs)
- Kubernetes operators for cloud-native deployment
- BB84 quantum key exchange protocol
- eIDAS 2.0 digital signature framework
- IBM Secure Enclave attestation integration
- Complete cryptographic bill of materials (CBOM)
- Hardware security module (HSM) bridging
- Multi-tenant key management system

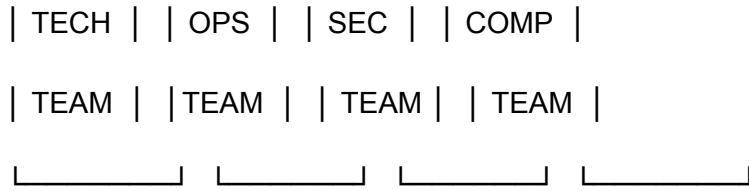
- Compliance monitoring and audit trails
  - Production Docker/Helm charts
  - Enterprise support and SLA commitments

## Out of Scope (Future Phases)

- X Quantum computing hardware deployment
  - X ML-KEM-512/768 variants (phase 2)
  - X Lattice-based signatures (Dilithium phase 2)
  - X Hardware quantum random number generator (post-2027)
  - X Custom ASIC cryptographic accelerators

## 1.2 Governance Structure





## 1.3 Regulatory Landscape

### eIDAS 2.0 Requirements (December 2024 - December 2026)

- Mandatory PQC algorithm support by Dec 31, 2026
- Digital signatures must use post-quantum algorithms
- Transitional hybrid mode allowed through 2026
- NIST-approved algorithms only (no proprietary variants)

### NIST Cryptographic Standards (2023-2024)

- FIPS 203: ML-KEM (Kyber) - Key encapsulation mechanism
- FIPS 204: ML-DSA (Dilithium) - Digital signatures (future)
- FIPS 205: SLH-DSA (SPHINCS+) - Stateless hash-based signatures (future)

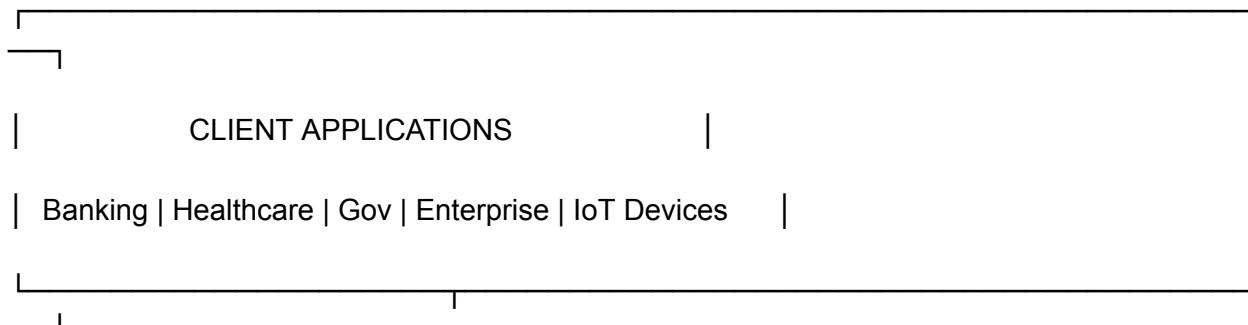
### Compliance Certifications

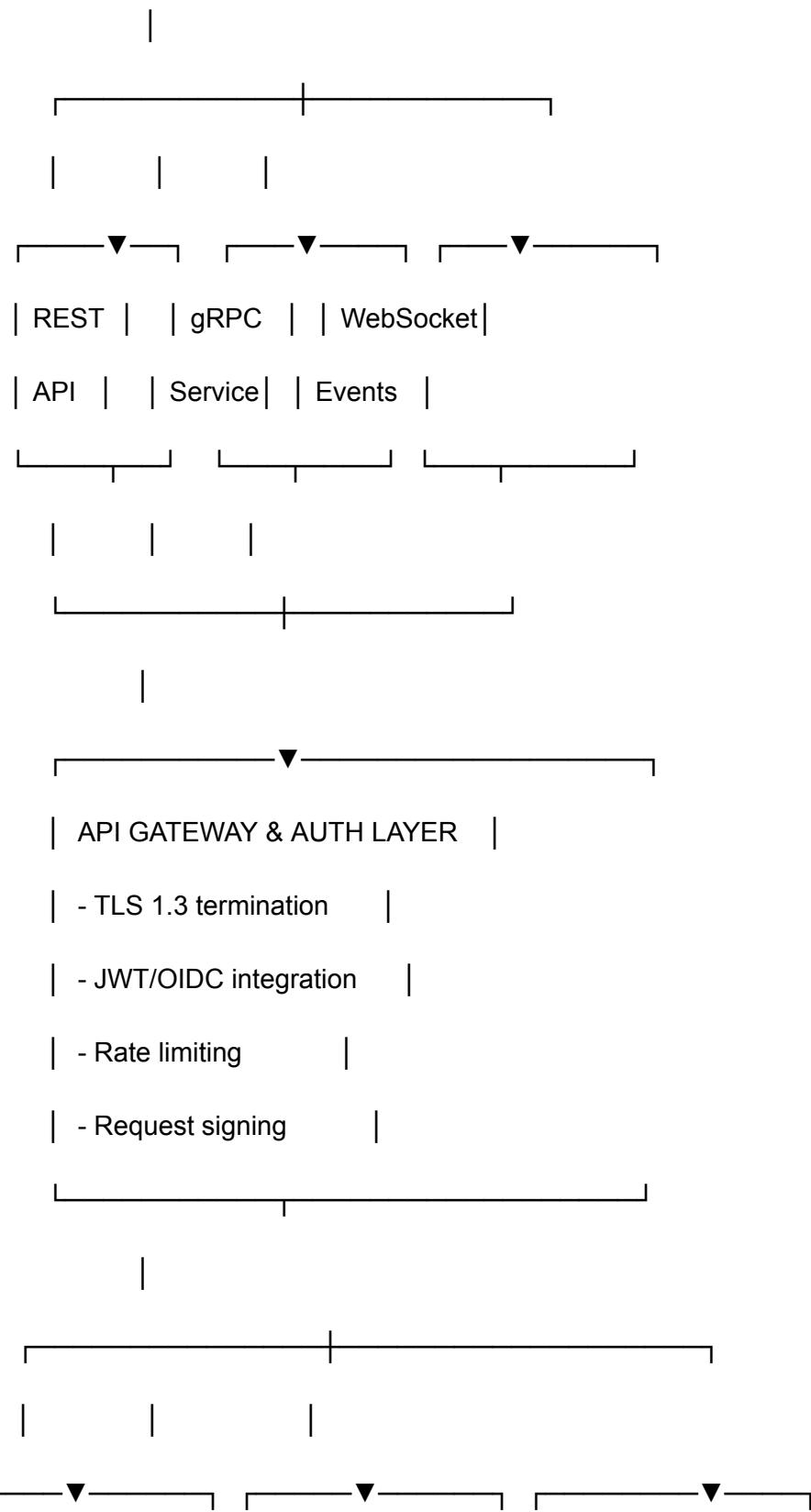
- SOC 2 Type II (security, availability, confidentiality)
- ISO 27001 (information security management)
- ISO 27002 (cryptographic controls)
- FIPS 140-3 (cryptographic module validation)

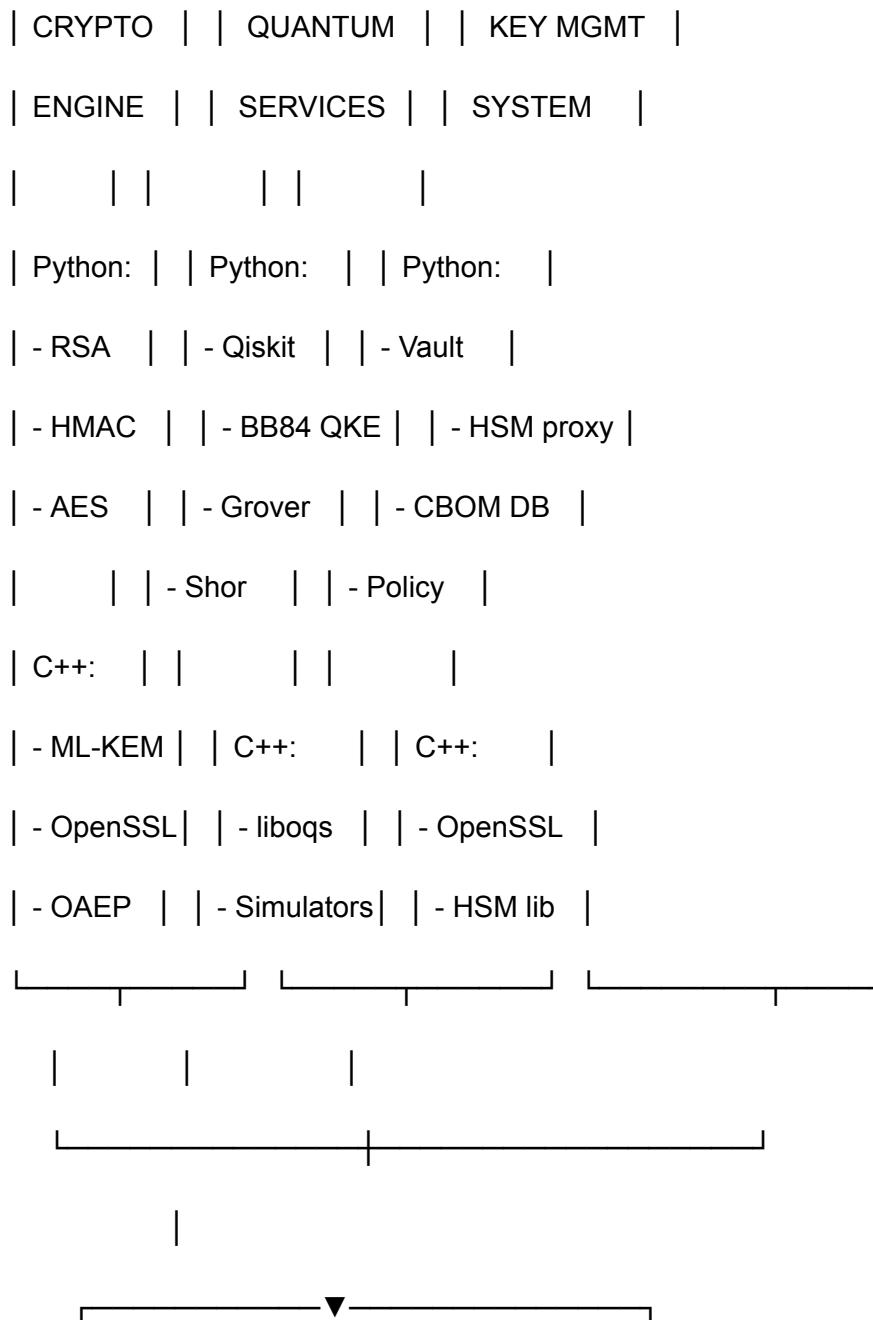
---

## SECTION 2: ARCHITECTURE OVERVIEW

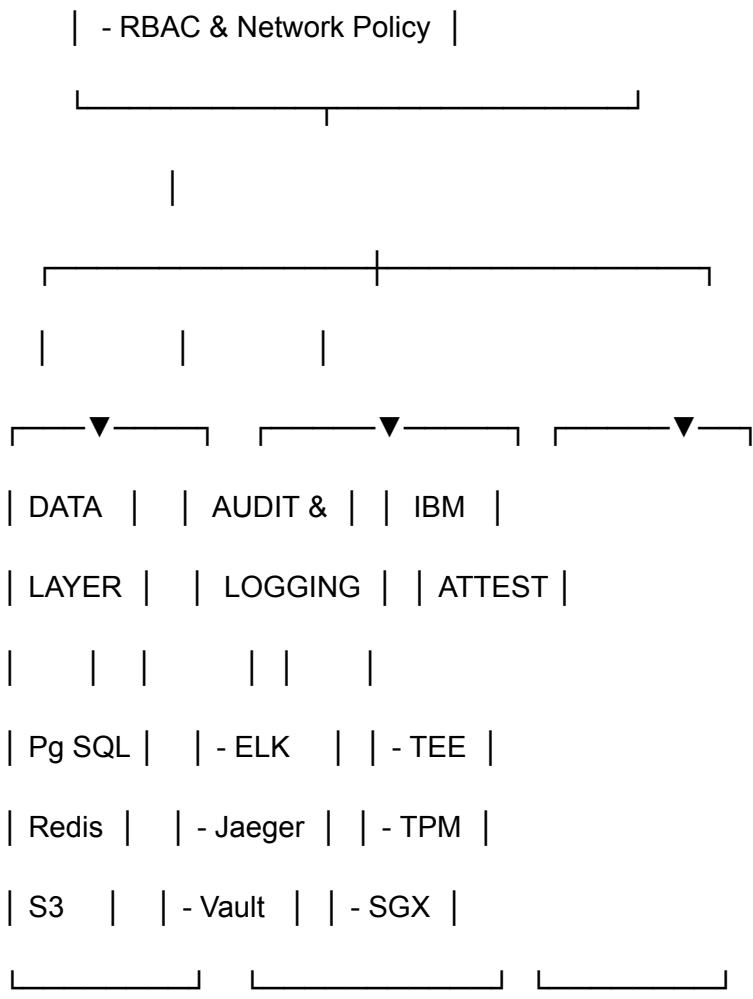
### 2.1 High-Level System Architecture







| KUBERNETES ORCHESTRATION |  
| - StatefulSets (DB) |  
| - Deployments (Services) |  
| - PersistentVolumes |  
| - ConfigMaps/Secrets |



## 2.2 Technology Stack

### Core Cryptographic Components

Component	Technology	Version	Role
<b>Key Encapsulation</b>	ML-KEM (Kyber)	NIST FIPS 203	Post-quantum KEX
<b>Classical KEM</b>	RSA-2048 + OAEP	PKCS#1 v2.2	Backward compatibility
<b>Quantum Simulation</b>	Qiskit	0.43.0	BB84 protocol, algorithm testing
<b>Quantum-Safe Lib</b>	liboqs	0.9.0	ML-KEM C bindings
<b>TLS Library</b>	OpenSSL	3.1.x	X.509, TLS 1.3

Component	Technology	Version	Role
<b>Crypto Backend</b>	pycryptodome	3.19.0	AES, HMAC, MD5 (legacy)

## Platform Infrastructure

Layer	Technology	Purpose
<b>Container Orchestration</b>	Kubernetes 1.28+	Cluster management, scheduling
<b>Service Mesh</b>	Istio 1.17+	Traffic management, mTLS, observability
<b>Package Management</b>	Helm 3.12+	Application deployment, templating
<b>Container Runtime</b>	containerd 1.7+	OCI-compliant runtime
<b>Storage</b>	PostgreSQL 15+ (data), Redis 7+ (cache)	Stateful services
<b>Messaging</b>	RabbitMQ 3.12+	Asynchronous operations, event streaming

## Security & Attestation

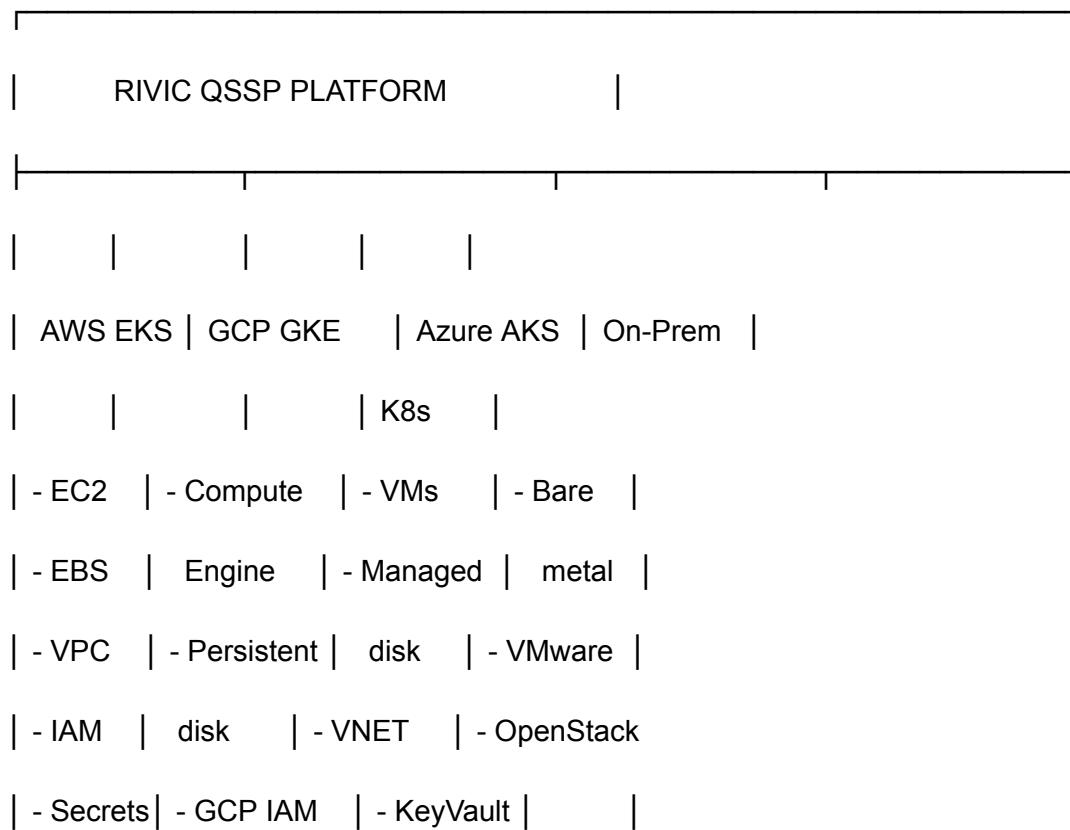
Component	Technology	Purpose
<b>Hardware Attestation</b>	IBM Secure Enclave / TPM 2.0	Hardware-backed verification
<b>Secret Management</b>	HashiCorp Vault 1.15+	Encrypted key storage
<b>Hardware Security Module</b>	PKCS#11 compatible HSM	Key ceremony, offline signing
<b>Compliance Monitoring</b>	Falco + OPA (Open Policy Agent)	Runtime security, policy enforcement
<b>Audit Logging</b>	Elasticsearch + Logstash + Kibana	Centralized audit trails

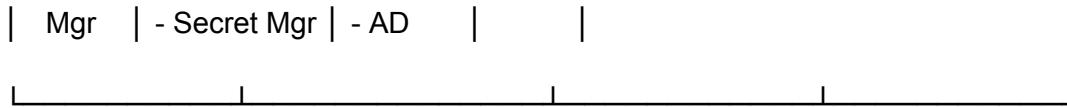
## DevOps & Operations

Component	Technology	Purpose
CI/CD Pipeline	GitLab CI / GitHub Actions	Automated testing, deployment
Container Registry	Docker Registry / ECR	Image storage and versioning
Monitoring	Prometheus + Grafana	Metrics collection, visualization
Distributed Tracing	Jaeger	Request tracing, latency analysis
Infrastructure as Code	Terraform	Cloud infrastructure provisioning

## 2.3 Deployment Model

### Multi-Cloud Support





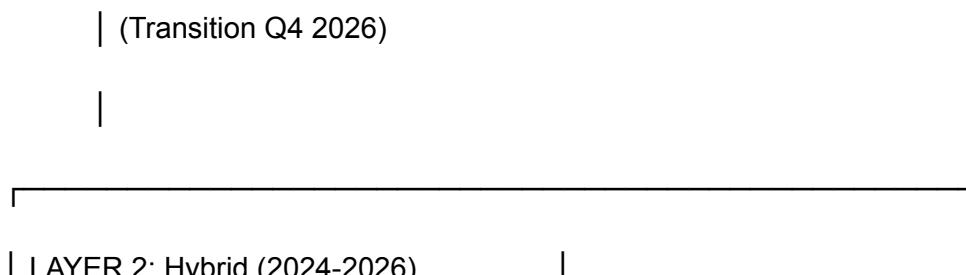
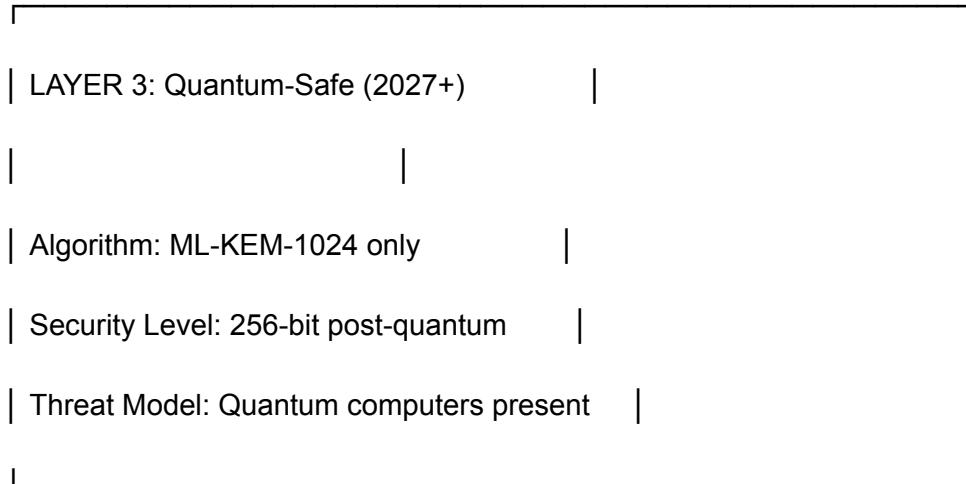
### Deployment Options:

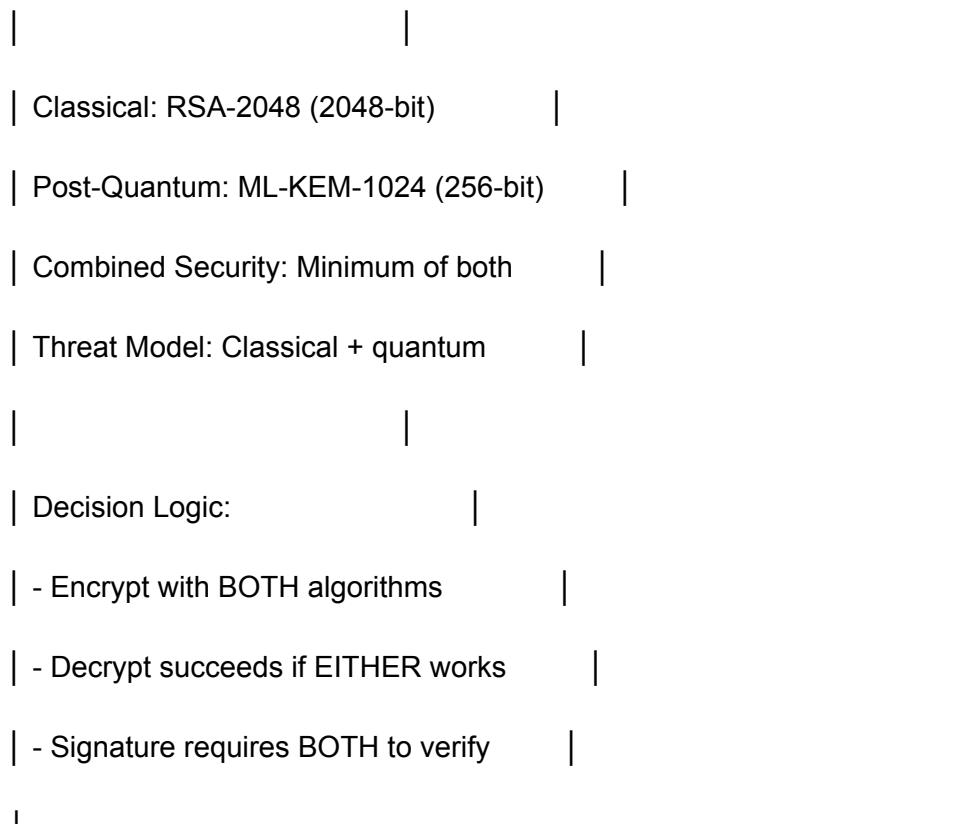
1. **SaaS** - Managed service hosted by Rivic
  2. **Dedicated Cloud** - Single-tenant AWS/GCP/Azure deployment
  3. **Hybrid** - On-prem control plane + cloud data plane
  4. **On-Premises** - Complete private deployment
- 

## SECTION 3: CRYPTOGRAPHIC ARCHITECTURE

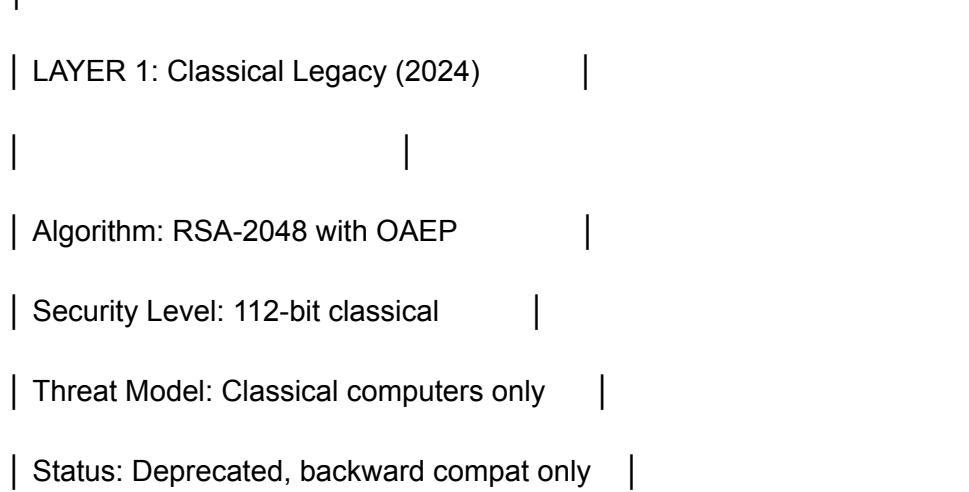
### 3.1 Hybrid Cryptography Model

#### Three-Layer Security Model





| (Deployment Q2 2025)



## Key Exchange Protocol Comparison

Protocol	Classical	Post-Quantum	Hybrid	Selection
RSA-KEM	✓ Yes	✗ No	Fallback	Legacy systems only
ECDH	✓ Yes	✗ No	Fallback	P-256, P-384 curves
ML-KEM-1024	✗ No	✓ Yes	Primary	NIST FIPS 203
ML-KEM-768	✗ No	✓ Yes	Future	Phase 2 (2027)
Hybrid (RSA+KEM)	✓ Yes	✓ Yes	✓ Yes	<b>Recommended</b>
BB84 QKE	✓ Simulated	✓ Quantum-ready	Future	2026+ with quantum HW

## 3.2 Cryptographic Key Lifecycle

### Key Generation, Distribution, and Rotation

#### 1. KEY GENERATION

- └─ Entity: Rivic PKI Service
- └─ Method: liboqs ML-KEM-1024 + OpenSSL RSA-2048
- └─ Entropy: /dev/urandom (kernel) + quantum seed
- └─ Storage: Secure enclave (IBM TEE)
- └─ Audit: CBOM entry created

#### 2. KEY CERTIFICATION

- └─ X.509 v3 certificate chain
- └─ Subject: Client/Service identity
- └─ Validity: 2-5 years (configurable)

- |—— Extensions: Enhanced key usage, policy constraints
- |—— CA: Rivic Root CA (offline) → Intermediate → Leaf

### 3. KEY DISTRIBUTION

- |—— Transport: TLS 1.3 with mutual auth
- |—— Recipient: Authenticated client or service
- |—— Protection: PQC-encrypted key material
- |—— Audit: Distribution logged with:
  - | |—— Recipient identity
  - | |—— Timestamp
  - | |—— Cryptographic algorithm used
  - | |—— Authorization decision
- |—— Backup: Sealed in Vault

### 4. KEY ESCROW & RECOVERY

- |—— Mechanism: Shamir secret sharing (3-of-5)
- |—— Participants: Authorized trustees (separate entities)
- |—— Activation: Requires majority consent + biometric
- |—— Recovery: Time-locked (7-day escrow minimum)
- |—— Audit: All escrow operations logged

### 5. KEY ROTATION

- |—— Trigger: Scheduled (annually) or on-demand
- |—— Method: Generate new key pair

- |—— Migration: Cryptographic agility (algorithm change)
- |—— Period: 30-day overlap between old and new keys
- |—— Re-encryption: Background job with audit trail
- └—— Deactivation: Old key archived, monitoring active

## 6. KEY RETIREMENT & ARCHIVAL

- |—— Retention: Compliance-dependent (7-30 years)
- |—— Storage: Cold storage (S3 Glacier), encrypted
- |—— Access: Quarterly audit of archived keys
- |—— Destruction: Cryptographic shredding (NIST guidelines)
- └—— Certificate: Revocation list (CRL) updated

### 3.3 Cryptographic Algorithm Selection Matrix

#### Data Classification & Algorithm Mapping

DATA CLASSIFICATION   ALGORITHM	
PUBLIC	<ul style="list-style-type: none"> <li>- No encryption required</li> <li>- Integrity: HMAC-SHA256</li> <li>- Non-repudiation: RSA signature</li> </ul>
INTERNAL	<ul style="list-style-type: none"> <li>- Encryption: RSA-2048 (legacy)</li> </ul>

- Integrity: HMAC-SHA256	
- Session: AES-256-GCM	
CONFIDENTIAL	
- Encryption: ML-KEM-1024	
- Integrity: HMAC-SHA256	
- Session: AES-256-GCM	
- Signing: RSA-4096 (until 2026)	
SECRET / FINANCIAL / HEALTH	
- Encryption: Hybrid (RSA + ML-KEM)	
- Integrity: HMAC-SHA256	
- Key derivation: HKDF-SHA256	
- Key wrapping: RFC 3394 (AES-KW)	
- Signing: Dual signature (2026)	
- Attestation: IBM TEE mandatory	
CLASSIFIED / STRATEGIC	
- Encryption: ML-KEM-1024 only	
- Integrity: HMAC-SHA512	
- Key agreement: Hybrid	

| - Signing: Multiple signatures |

| - Attestation: Hardware verify |

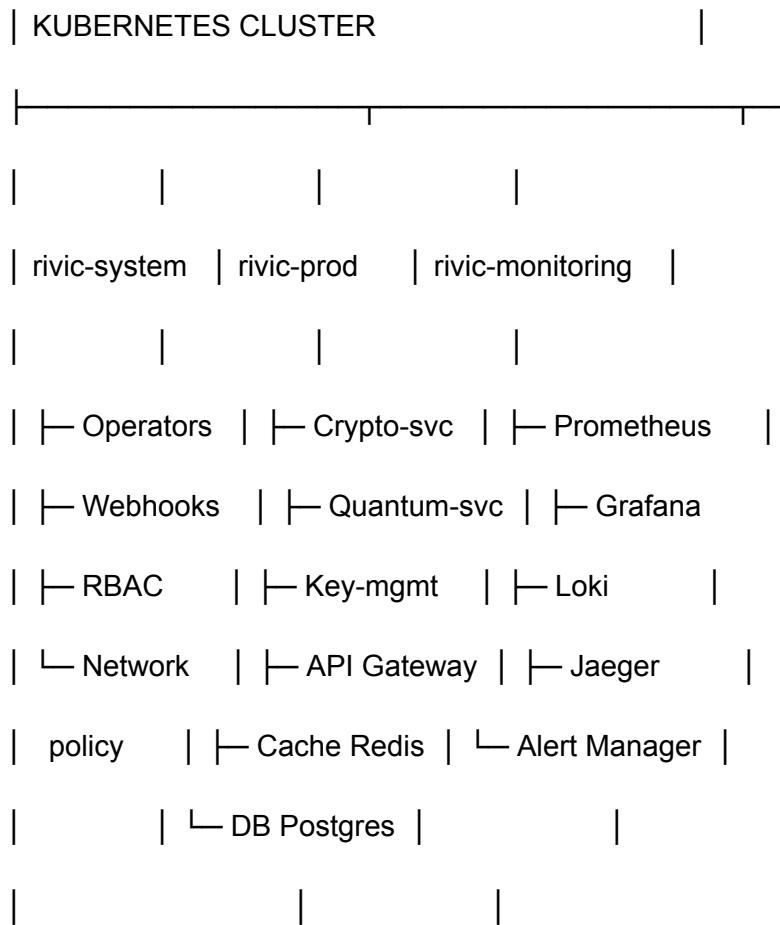
| - Audit: Immutable blockchain log |

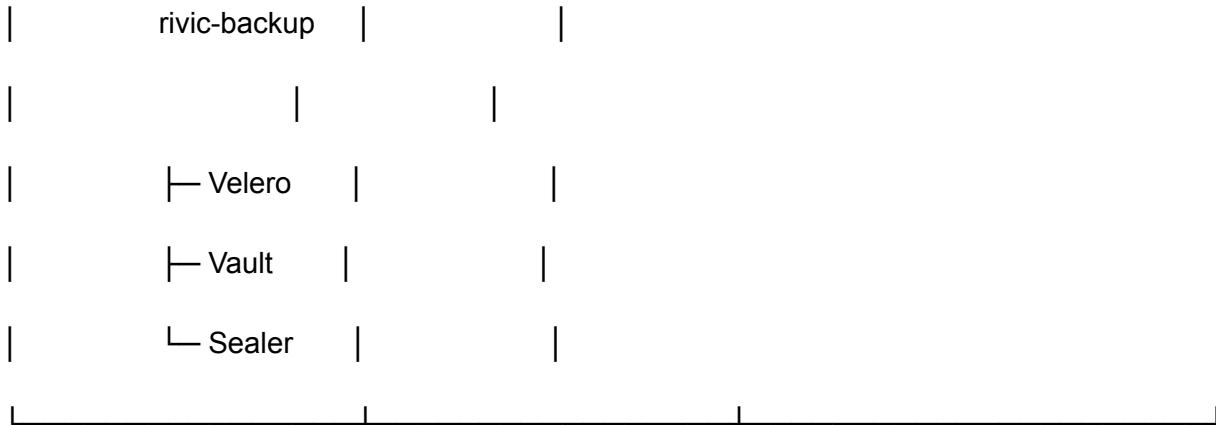
---

## SECTION 4: CLOUD-NATIVE ARCHITECTURE

### 4.1 Kubernetes Deployment Architecture

#### Namespace and Workload Organization





## Service Mesh Integration (Istio)

```
# Istio mTLS enforcement
```

```
apiVersion: security.istio.io/v1beta1
```

```
kind: PeerAuthentication
```

```
metadata:
```

```
  namespace: rivic-prod
```

```
spec:
```

```
  mtls:
```

```
    mode: STRICT # Enforce mTLS for all services
```

```
    privateKey:
```

```
      # PQC-enabled certificate generation
```

```
      algorithm: "ML-KEM-1024"
```

```
      size: 1568 # bytes
```

```
---
```

```
# Service authorization policy
```

```
apiVersion: security.istio.io/v1beta1
```

```
kind: AuthorizationPolicy

metadata:
  namespace: rivic-prod

spec:
  selector:
    matchLabels:
      app: quantum-service

  action: ALLOW

  rules:
    - from:
        - source:
            principals: ["cluster.local/ns/rivic-prod/sa/crypto-engine"]

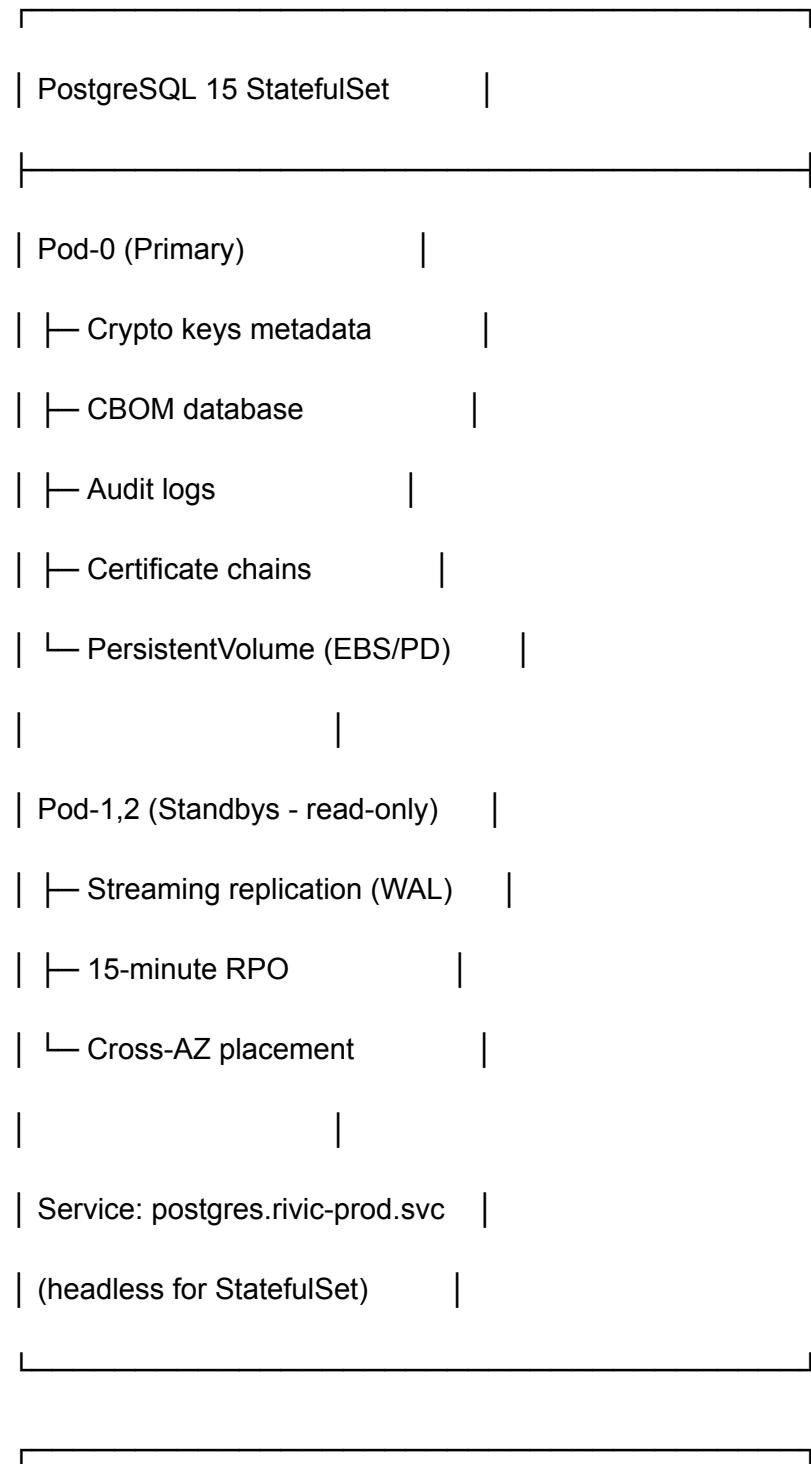
        to:
          - operation:
              methods: ["POST"]
              paths: ["/v1/encrypt", "/v1/decrypt"]

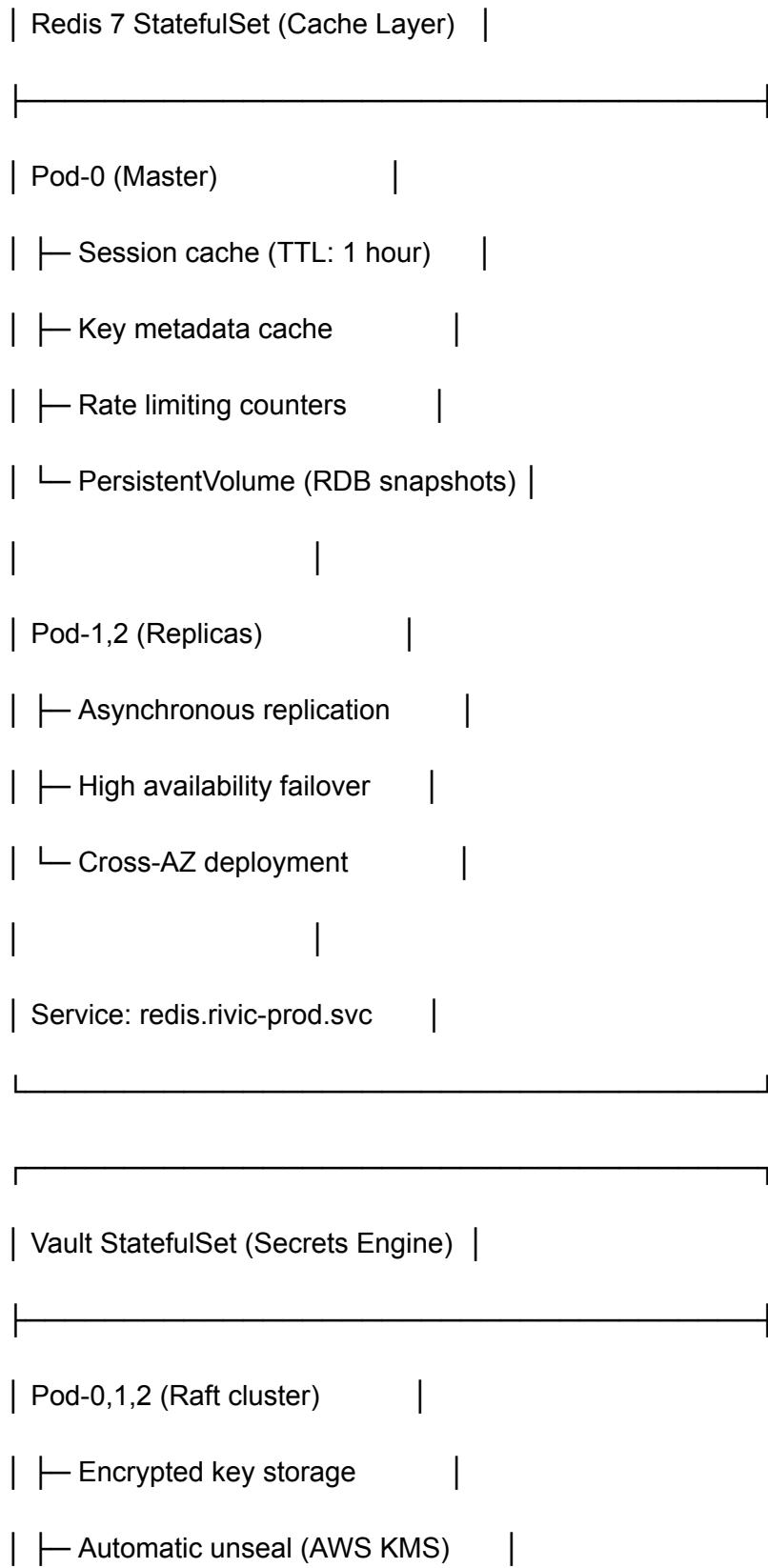
        - from:
            - source:
                principals: ["cluster.local/ns/rivic-prod/sa/api-gateway"]

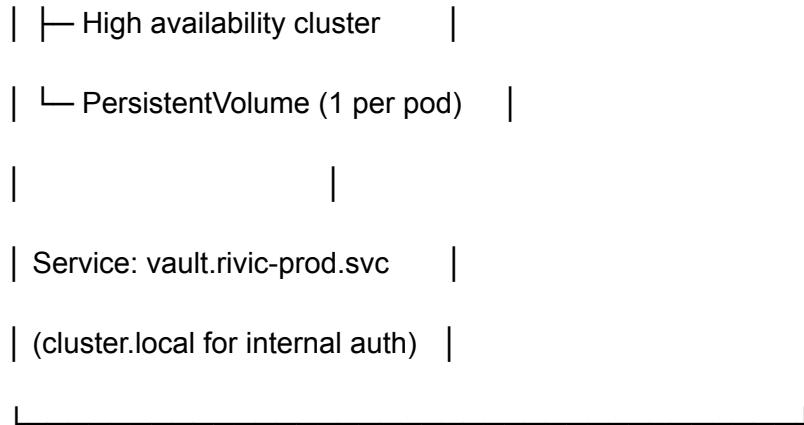
            to:
              - operation:
```

```
methods: ["POST"]  
paths: ["/v1/qke/generate"]
```

## Stateful Service Architecture







## 4.2 Cloud-Native Cryptographic Patterns

### Pattern 1: Cryptographic Sidecar Injection

```
apiVersion: v1
```

```
kind: Pod
```

```
metadata:
```

```
annotations:
```

```
# Injected automatically by webhook
```

```
rivic.io/inject-crypto-sidecar: "true"
```

```
rivic.io/crypto-algorithm: "ML-KEM-1024"
```

```
spec:
```

```
containers:
```

```
- name: app-container
```

```
image: myapp:1.0
```

```
volumeMounts:
```

```
- name: crypto-socket
```

```
mountPath: /var/run/rivic-crypto
```

env:

- name: RIVIC\_CRYPTO\_ENDPOINT  
value: unix:///var/run/rivic-crypto/crypto.sock

# Injected sidecar

- name: crypto-sidecar  
image: rivic/crypto-sidecar:1.0  
securityContext:  
privileged: true # For TPM access

volumeMounts:

- name: crypto-socket  
mountPath: /var/run/rivic-crypto
- name: tpm-device  
mountPath: /dev/tpm0

volumes:

- name: crypto-socket  
emptyDir: {}
- name: tpm-device  
hostPath:  
path: /dev/tpm0

## **Pattern 2: Cryptographic Initiation Sidecar**

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
  name: secure-app
```

```
spec:
```

```
  template:
```

```
    spec:
```

```
      initContainers:
```

```
        # Initialize cryptographic environment
```

```
        - name: crypto-init
```

```
          image: rivic/crypto-init:1.0
```

```
          env:
```

```
            - name: RIVIC_KEY_NAME
```

```
              value: app-signing-key
```

```
            - name: RIVIC_ALGORITHM
```

```
              value: ML-KEM-1024
```

```
          volumeMounts:
```

```
            - name: crypto-volume
```

```
              mountPath: /etc/rivic-keys
```

```
        # Runs to completion before app container starts
```

```
  securityContext:  
    runAsUser: 0  
  
  capabilities:  
    add:  
      - SYS_ADMIN # For attestation
```

```
  containers:  
    - name: app  
      image: myapp:1.0  
  
  volumeMounts:  
    - name: crypto-volume  
      mountPath: /etc/rivic-keys  
      readOnly: true
```

```
  volumes:  
    - name: crypto-volume  
      emptyDir:  
        sizeLimit: 1Gi
```

### **Pattern 3: Cryptographic Access Control**

```
# Network Policy: Restrict crypto service access
```

```
apiVersion: networking.k8s.io/v1
```

```
kind: NetworkPolicy
```

```
metadata:
```

```
namespace: rivic-prod
```

```
spec:
```

```
podSelector:
```

```
matchLabels:
```

```
    app: crypto-service
```

```
policyTypes:
```

```
    - Ingress
```

```
    - Egress
```

```
ingress:
```

```
    - from:
```

```
        # Only authorized services
```

```
        - podSelector:
```

```
            matchLabels:
```

```
                role: authorized-client
```

```
ports:
```

```
    - protocol: TCP
```

```
        port: 9443 # TLS port
```

```
egress:
```

```
    # Allow outbound only to Vault
```

```
- to:  
  - podSelector:  
    matchLabels:  
      app: vault  
    ports:  
      - protocol: TCP  
        port: 8200  
---  
# RBAC: Service account authorization  
  
apiVersion: v1  
  
kind: ServiceAccount  
  
metadata:  
  name: crypto-engine  
---  
apiVersion: rbac.authorization.k8s.io/v1  
  
kind: ClusterRole  
  
metadata:  
  name: crypto-engine-role  
  
rules:  
  - apiGroups: [""]  
    resources: ["secrets"]  
    verbs: ["get", "list"]
```

```
resourceNames: ["crypto-keys"] # Restrict to specific secret

- apiGroups: [""]

resources: ["configmaps"]

verbs: ["get"]

resourceNames: ["crypto-config"]

---

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: crypto-engine-binding

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: crypto-engine-role

subjects:

- kind: ServiceAccount

name: crypto-engine

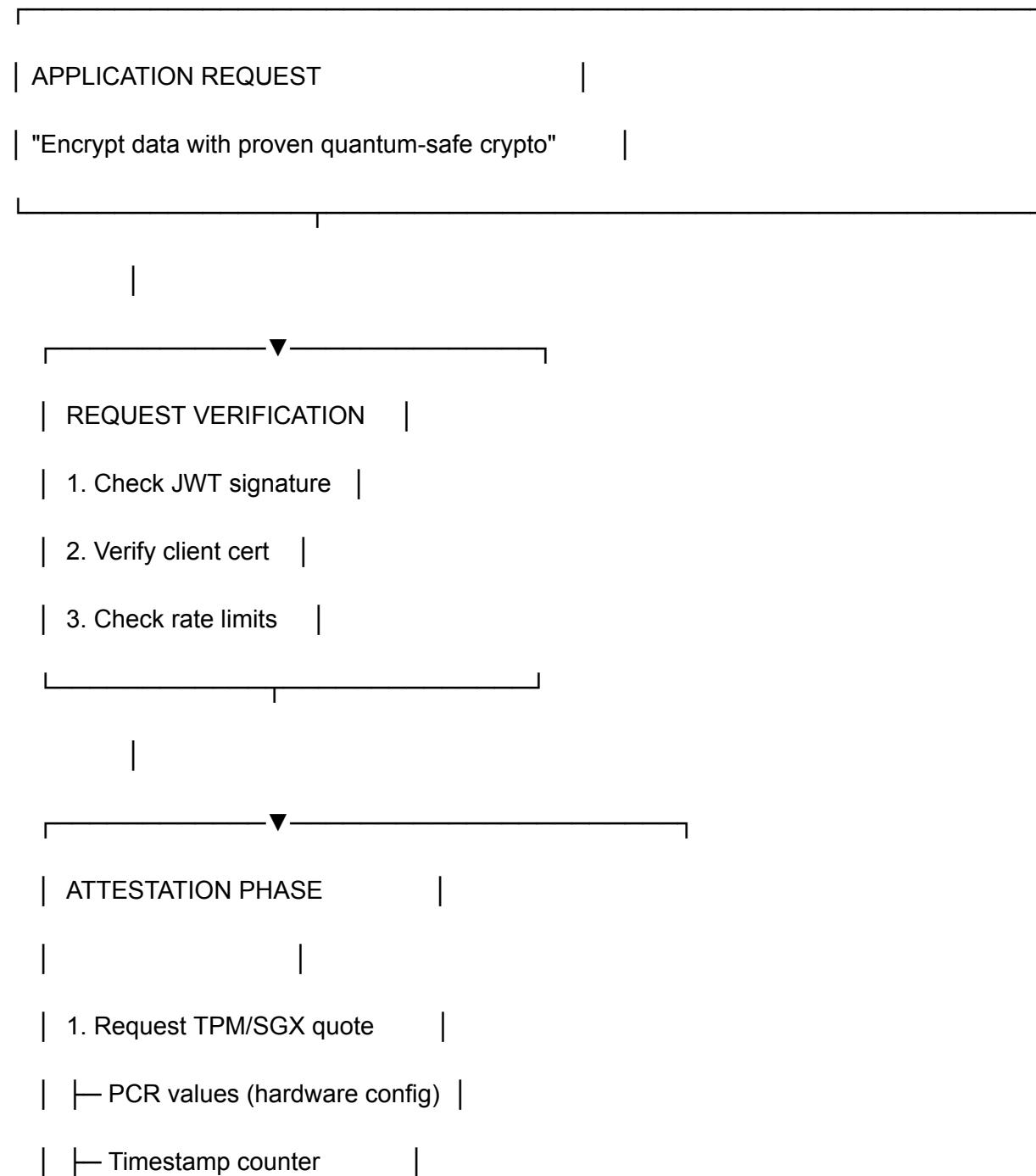
namespace: rivic-prod
```

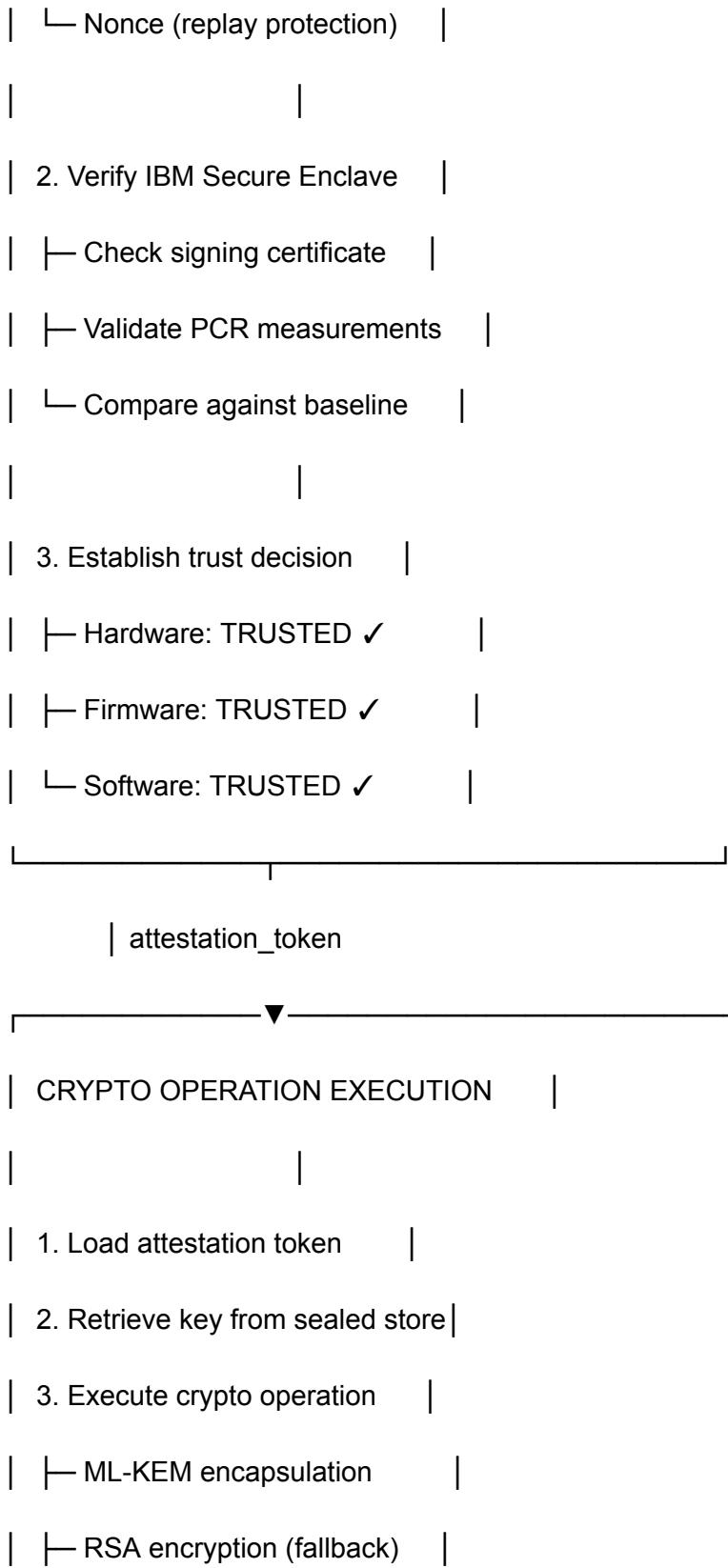
---

## SECTION 5: IBM ATTESTATION INTEGRATION

### 5.1 Attestation Architecture

#### Hardware Attestation Flow





```
| └─ HMAC-SHA256 (integrity)   |
| 4. Clear sensitive data from RAM |
| 5. Return encrypted result    |
└─────────────────────────────────
```

## AUDIT & COMPLIANCE LOGGING

```
|           |
| Record:      |
| - Requestor identity   |
| - Timestamp (UTC)     |
| - Algorithm used       |
| - Attestation result   |
| - Operation status     |
| - Hardware PCR values |
```

```
|           |
| Storage:      |
| - Immutable audit log (append) |
| - Encrypted in storage        |
| - Replicated to cold storage |
```

## 5.2 IBM Secure Enclave Integration

### Trusted Execution Environment (TEE) Setup

# Kubernetes deployment with IBM attestation

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
  name: crypto-engine-attestable
```

```
spec:
```

```
  template:
```

```
    metadata:
```

```
      annotations:
```

```
        # Enable IBM attestation
```

```
        ibm.com/attestation-enabled: "true"
```

```
        ibm.com/enclave-type: "secure_enclave"
```

```
    spec:
```

```
      # Required for TEE operation
```

```
      affinity:
```

```
        nodeAffinity:
```

```
          requiredDuringSchedulingIgnoredDuringExecution:
```

```
            nodeSelectorTerms:
```

```
              - matchExpressions:
```

```
- key: node.kubernetes.io/enclave-capable
  operator: In
  values: ["true"]
```

containers:

```
- name: crypto-engine
  image: rivic/crypto-engine:1.0-attestable
```

securityContext:

```
# Required for TPM 2.0 access
```

```
privileged: true
```

```
allowPrivilegeEscalation: true
```

volumeMounts:

```
- name: tpm-device
  mountPath: /dev/tpm0
```

```
- name: attestation-certs
```

```
  mountPath: /etc/attestation
```

```
  readOnly: true
```

env:

```
# IBM attestation configuration
```

```
- name: IBM_ATTESTATION_ENABLED
```

```
  value: "true"
```

```
- name: IBM_ATTESTATION_SERVER
```

```
    value: "attestation.us-south.cloud.ibm.com"

- name: IBM_ATTESTATION_CERT_PATH

    value: "/etc/attestation/ibm-enclave.pem"

- name: CRYPTO_ALGORITHM

    value: "ML-KEM-1024"

- name: ATTESTATION_INTERVAL_SECONDS

    value: "3600" # Re-attest hourly
```

volumes:

```
- name: tpm-device
```

hostPath:

```
  path: /dev/tpm0
```

```
  type: CharDevice
```

```
- name: attestation-certs
```

secret:

```
  secretName: ibm-attestation-certs
```

```
  defaultMode: 0400 # Read-only
```

## Attestation Verification Code (Python)

\*\*\*\*

IBM Secure Enclave Attestation Integration

Verifies hardware and software integrity before crypto operations

"""

```
from dataclasses import dataclass

from typing import Dict, Optional

import requests

import json

import logging

from datetime import datetime, timedelta

import hashlib

import struct

logger = logging.getLogger(__name__)

@dataclass

class AttestationQuote:

    """IBM TEE attestation quote"""

    quote_data: bytes

    signature: bytes

    pcr_values: Dict[int, bytes]

    timestamp: datetime

    nonce: bytes

    enclave_type: str # "secure_enclave", "tpm2", "sgx"

class IBMAttestationVerifier:

    """
```

Verify IBM Secure Enclave attestation for crypto operations

"""

```
def __init__(self,  
            attestation_server: str = "attestation.us-south.cloud.ibm.com",  
            cert_path: str = "/etc/attestation/ibm-enclave.pem"):  
  
    self.attestation_server = attestation_server  
  
    self.cert_path = cert_path  
  
    self.cached_policies: Dict[str, dict] = {}  
  
    self.last_policy_update = None  
  
    self._load_root_certificate()
```

```
def _load_root_certificate(self):
```

"""Load IBM root CA certificate"""

try:

```
        with open(self.cert_path, 'rb') as f:  
            self.ibm_root_cert = f.read()  
  
        logger.info(f"Loaded IBM root certificate from {self.cert_path}")
```

except FileNotFoundError:

```
        logger.error(f"IBM attestation certificate not found: {self.cert_path}")
```

```
        raise
```

```
def generate_attestation_request(self,  
                                 algorithm: str = "ML-KEM-1024") -> Dict:  
    """
```

Generate attestation request with nonce

Args:

algorithm: Cryptographic algorithm to attest

Returns:

Attestation request with nonce and timestamp

"""

```
import secrets
```

```
nonce = secrets.token_bytes(32)
```

```
timestamp = datetime.utcnow()
```

```
return {
```

"nonce": nonce.hex(),

"timestamp": timestamp.isoformat(),

"algorithm": algorithm,

"pcr\_mask": 0x0FFF, # Request PCRs 0-11

```
        "enclave_type": "secure_enclave"

    }

def verify_attestation_quote(self,
                             quote: AttestationQuote,
                             expected_nonce: bytes,
                             trust_baseline: Dict[int, str]) -> bool:
```

.....

Verify IBM TEE attestation quote

Args:

```
    quote: Attestation quote from TEE
    expected_nonce: Expected nonce (replay protection)
    trust_baseline: Trusted PCR values by index
```

Returns:

True if attestation verified, False otherwise

.....

```
# 1. Verify quote signature

logger.info("Verifying quote signature...")

if not self._verify_quote_signature(quote):
```

```
logger.error("Quote signature verification failed")

return False

# 2. Check nonce (replay protection)

logger.info("Checking nonce...")

if quote.nonce != expected_nonce:
    logger.error(f"Nonce mismatch: expected {expected_nonce.hex()}, "
                f"got {quote.nonce.hex()}")

return False

# 3. Verify timestamp is recent

logger.info("Checking timestamp freshness... ")

age = datetime.utcnow() - quote.timestamp

if age > timedelta(minutes=5):
    logger.warning(f"Quote is {age.total_seconds()} seconds old, "
                  "exceeds freshness threshold")

return False

# 4. Verify PCR values against baseline

logger.info("Comparing PCR values against baseline...")

for pcr_index, expected_value in trust_baseline.items():
```

```
actual_value = quote.pcr_values.get(pcr_index)

if actual_value is None:
    logger.error(f"PCR {pcr_index} not present in quote")
    return False

actual_hex = actual_value.hex()

if actual_hex != expected_value:
    logger.error(
        f"PCR {pcr_index} mismatch:\n"
        f" Expected: {expected_value}\n"
        f" Got:     {actual_hex}"
    )
    return False

# 5. Verify with IBM attestation server
logger.info("Verifying with IBM attestation service...")
try:
    response = requests.post(
        f"https://{{self.attestation_server}}/verify",
        json={
            "quote": quote.quote_data.hex(),

```

```
        "signature": quote.signature.hex(),

        "certificate_chain": self.ibm_root_cert.decode('utf-8'),

    },

    timeout=10,

    verify=True

)

if response.status_code != 200:

    logger.error(f"IBM attestation service returned {response.status_code}")

    return False

result = response.json()

if not result.get("verified", False):

    logger.error(f"IBM verification failed: {result.get('reason')}")

    return False

except requests.RequestException as e:

    logger.error(f"Failed to contact IBM attestation service: {e}")

    return False

logger.info("✓ All attestation checks passed")
```

```
return True

def _verify_quote_signature(self, quote: AttestationQuote) -> bool:
    """
    Verify RSA signature on attestation quote

    Uses IBM's public key from root certificate
    """

    from cryptography import x509
    from cryptography.hazmat.backends import default_backend
    from cryptography.hazmat.primitives import hashes
    from cryptography.hazmat.primitives.asymmetric import padding

    try:
        cert = x509.load_pem_x509_certificate(
            self.ibm_root_cert,
            default_backend()
        )
        public_key = cert.public_key()

        public_key.verify(
            quote.signature,
            quote.quote_data,
        )
    except Exception as e:
        logger.error(f"Failed to verify quote signature: {e}")
        return False

    return True
```

```
padding.PKCS1v15(),  
  
hashes.SHA256()  
  
)  
  
return True  
  
except Exception as e:  
  
    logger.error(f"Signature verification failed: {e}")  
  
return False  
  
  
  
def execute_attested_crypto_operation(self,  
  
                                      operation: str,  
  
                                      data: bytes,  
  
                                      public_key: bytes,  
  
                                      algorithm: str = "ML-KEM-1024") -> Optional[bytes]:  
  
    """  
  
Execute cryptographic operation only after successful attestation
```

Args:

operation: "encrypt" or "decrypt"  
data: Data to encrypt/decrypt  
public\_key: Public key for operation  
algorithm: Crypto algorithm to use

Returns:

Encrypted/decrypted data if attestation succeeds, None otherwise

"""

# Step 1: Generate attestation request

```
attest_request = self.generate_attestation_request(algorithm)
```

# Step 2: Get quote from TEE

```
quote = self._get_tee_quote(attest_request)
```

# Step 3: Verify attestation

```
trust_baseline = self._get_trust_baseline(algorithm)
```

```
if not self.verify_attestation_quote(
```

```
    quote,
```

```
    bytes.fromhex(attest_request["nonce"]),

```

```
    trust_baseline

```

```
):
```

```
    logger.error("Attestation failed - operation denied")

```

```
    return None

```

# Step 4: Execute crypto operation (in TEE-protected memory)

```
logger.info(f"Attestation successful - executing {operation}...")  
  
if operation == "encrypt":  
    result = self._execute_pqc_encrypt(data, public_key, algorithm)  
  
elif operation == "decrypt":  
    result = self._execute_pqc_decrypt(data, public_key, algorithm)  
  
else:  
    raise ValueError(f"Unknown operation: {operation}")  
  
# Step 5: Log operation in audit trail  
  
self._audit_log_crypto_operation(  
    operation=operation,  
    algorithm=algorithm,  
    attestation_result="PASSED",  
    pcr_values=quote.pcr_values  
)  
  
return result
```

```
def _get_tee_quote(self, request: Dict) -> AttestationQuote:  
    """Get attestation quote from TEE (TPM/SGX/Secure Enclave)"""
```

```
# In real implementation, would use libtss2 or sgx-quote-provider
logger.info("Requesting quote from TEE...")

# Placeholder - would call actual TEE library

import secrets

from datetime import datetime

return AttestationQuote(
    quote_data=secrets.token_bytes(1024),
    signature=secrets.token_bytes(256),
    pcr_values={
        0: hashlib.sha256(b"firmware_config").digest(),
        1: hashlib.sha256(b"config_data").digest(),
        7: hashlib.sha256(b"secure_boot_config").digest(),
    },
    timestamp=datetime.utcnow(),
    nonce=bytes.fromhex(request["nonce"]),
    enclave_type="secure_enclave"
)

def _get_trust_baseline(self, algorithm: str) -> Dict[int, str]:
    """Get trusted PCR baseline for comparison"""

```

```
    if algorithm == "ML-KEM-1024":  
        k = 1024  
        n = 1024  
        p = 1024  
        q = 1024  
        r = 1024  
        s = 1024  
        t = 1024  
        u = 1024  
        v = 1024  
        w = 1024  
        x = 1024  
        y = 1024  
        z = 1024
```

return baseline

```
def _execute_pqc_encrypt(self, data: bytes, public_key: bytes,
                        algorithm: str) -> bytes:
    """Execute PQC encryption in attestation context"""
    from src.classical.kyber_operations import ML_KEM_Operations
```

```
if algorithm == "ML-KEM-1024":  
    kem = ML_KEM_Operations("ML-KEM-1024")  
  
    ciphertext, shared_secret = kem.encapsulate(public_key)  
  
    return ciphertext + shared_secret
```

```
else:
    raise ValueError(f"Unsupported algorithm: {algorithm}")

def _execute_pqc_decrypt(self, data: bytes, secret_key: bytes,
                        algorithm: str) -> Optional[bytes]:
    """Execute PQC decryption in attestation context"""
    from src.classical.kyber_operations import ML_KEM_Operations

    if algorithm == "ML-KEM-1024":
        kem = ML_KEM_Operations("ML-KEM-1024")
        # Split ciphertext and shared secret
        ct_size = 1568
        ciphertext = data[:ct_size]
        result = kem.decapsulate(ciphertext, secret_key)
        return result
    else:
        raise ValueError(f"Unsupported algorithm: {algorithm}")

def _audit_log_crypto_operation(self, **kwargs):
    """Log cryptographic operation to immutable audit trail"""
    audit_entry = {
        "timestamp": datetime.utcnow().isoformat(),
```

```
"event_type": "crypto_operation",

"request_id": kwargs.get("operation"),

"algorithm": kwargs.get("algorithm"),

"attestation_result": kwargs.get("attestation_result"),

"pcr_values": {

    k: v.hex() for k, v in kwargs.get("pcr_values", {}).items()

}

}

logger.info(f"AUDIT: {json.dumps(audit_entry)}")

# Would also write to encrypted audit log in database
```

---

## SECTION 6: CRYPTOGRAPHIC BILL OF MATERIALS (CBOM)

### 6.1 CBOM Schema and Structure

```
{
```

```
    "document": {

        "version": "2.0",

        "name": "Rivic QSSP Crypto BOM",

        "serialNumber": "urn:uuid:7d0f5d29-e3c0-4f2e-9e22-8b5f9e8c1234",

        "creationDate": "2025-12-26T14:30:00Z",

        "creationTool": "Rivic CBOM Generator v2.0"
```

```
    },  
  
    "metadata": {  
        "organization": "Rivic Inc.",  
        "contactEmail": "security@rivic.io",  
        "complianceFramework": "eIDAS 2.0, NIST SP 800-175B",  
        "lastUpdated": "2025-12-26T14:30:00Z"  
    },  
  
    "cryptographicElements": [  
        {  
            "id": "crypto-001",  
            "component": "ML-KEM-1024",  
            "category": "KeyEncapsulationMechanism",  
            "algorithm": {  
                "name": "ML-KEM-1024",  
                "specification": "NIST FIPS 203",  
                "status": "APPROVED",  
                "standardizationBody": "NIST",  
                "publicationDate": "2024-08-13",  
                "deprecationDate": null,  
                "retirementDate": null  
            }  
        }  
    ]  
}
```

```
    },  
  
    "keyLength": 1568,  
  
    "securityStrength": 256,  
  
    "quantumSafe": true,  
  
    "quantumResistance": "post-quantum",  
  
    "usage": [  
  
        "keyEncapsulation",  
  
        "keyDerivation",  
  
        "secureChannelEstablishment"  
  
    ],  
  
    "implementation": {  
  
        "library": "liboqs",  
  
        "version": "0.9.0",  
  
        "sourceUrl": "https://github.com/open-quantum-safe/liboqs",  
  
        "license": "MIT",  
  
        "certificateStatus": "pending-fips-validation"  
  
    },  
  
    "deployment": {  
  
        "platforms": ["Linux x86_64", "Kubernetes", "Cloud"],  
  
        "hardwareRequired": false,  
  
        "acceleratorUsed": false,  
  
        "containerized": true,  
  
    }  
}
```

```
"attestationRequired": true  
},  
  
"certificates": [  
    {  
        "certificateName": "FIPS 140-3 (Module #4387)",  
        "issuer": "NIST",  
        "issueDate": "2024-08-15",  
        "validUntil": "2029-08-15",  
        "level": 3,  
        "scope": "Post-quantum key encapsulation"  
    }  
,  
    "vulnerabilities": [],  
    "auditTrail": {  
        "deploymentDate": "2025-12-20T10:00:00Z",  
        "lastTestedDate": "2025-12-25T16:45:00Z",  
        "nextAuditDate": "2026-03-20T10:00:00Z"  
    }  
},  
  
{
```

```
"id": "crypto-002",  
  "component": "RSA-2048",  
  "category": "KeyEncryptionKey",  
  "algorithm": {  
    "name": "RSA",  
    "keySize": 2048,  
    "specification": "PKCS#1 v2.2",  
    "status": "DEPRECATED",  
    "deprecationDate": "2024-12-31",  
    "retirementDate": "2026-12-31",  
    "reason": "Vulnerable to quantum computer attacks post-2030"  
  },  
  "keyLength": 2048,  
  "securityStrength": 112,  
  "quantumSafe": false,  
  "quantumResistance": "vulnerable",  
  "usage": [  
    "keyWrap",  
    "digitalSignature",  
    "backward-compatibility-only"  
  ],  
  "implementation": {
```

```
        "library": "OpenSSL",
        "version": "3.1.4",
        "sourceUrl": "https://www.openssl.org",
        "license": "Apache 2.0"
    },
    "deploymentPhase": "DEPRECATION",
    "retirementPlan": {
        "phase1": "Parallel deployment (RSA + ML-KEM)",
        "phase1Dates": "2025-01-01 to 2026-12-31",
        "phase2": "Transition period (ML-KEM only)",
        "phase2Dates": "2027-01-01 to 2027-12-31",
        "phase3": "RSA key destruction",
        "phase3Date": "2027-12-31"
    },
    {
        "id": "crypto-003",
        "component": "HMAC-SHA256",
        "category": "MessageAuthenticationCode",
        "algorithm": {

```

```
        "name": "HMAC-SHA256",
        "specification": "RFC 4868",
        "status": "APPROVED",
        "quantumSafe": false,
        "requiredQuantumSafeAlternative": "KMAC256"
    },
    "keyLength": 256,
    "securityStrength": 256,
    "usage": ["integrityProtection", "authentication"],
    "implementation": {
        "library": "pycryptodome",
        "version": "3.19.0"
    },
    "replacementPlan": {
        "currentAlgorithm": "HMAC-SHA256",
        "futureAlgorithm": "KMAC256",
        "transitionDate": "2027-01-01",
        "riskAssessment": "low"
    }
},
{

```

```
"id": "crypto-004",

"component": "AES-256-GCM",

"category": "SymmetricEncryption",

"algorithm": {

    "name": "AES-256-GCM",

    "specification": "NIST SP 800-38D",

    "status": "APPROVED",

    "quantumSafe": false,

    "quantumSafeAlternative": "AES-256-GCM (with increased key length post-2030)"

},

"keyLength": 256,

"ivLength": 96,

"securityStrength": 256,

"usage": ["confidentiality", "authentication"],

"performance": {

    "throughput": "5-10 Gbps",

    "latency": "<100 microseconds",

    "hardwareAcceleration": "AES-NI, AVX-2"

}

},

],
```

```
"cryptographicComponents": {  
    "totalComponents": 4,  
    "approved": 3,  
    "deprecated": 1,  
    "vulnerable": 0,  
    "quantumSafe": 1,  
    "quantumVulnerable": 3  
},  
  
"complianceStatus": {  
    "eIDAS20": "READY",  
    "nistFIPS": "READY",  
    "nisppComplianceLevel": 3,  
    "tlsVersion": "1.3",  
    "certificateValidation": "X.509 v3 with quantum-safe algorithm"  
},  
  
"recommendations": [  
    {  
        "priority": "CRITICAL",  
        "issue": "RSA-2048 deprecated by NIST",  
        "action": "Complete migration to ML-KEM-1024 by 2026-12-31",  
        "estimatedEffort": "120 developer-hours",  
        "riskIfNotDone": "Non-compliance with eIDAS 2.0 after 2026"  
    }  
]
```

```
    },  
  
    {  
  
        "priority": "HIGH",  
  
        "issue": "HMAC-SHA256 non-quantum-safe",  
  
        "action": "Evaluate KMAC256 migration for post-2027 deployments",  
  
        "estimatedEffort": "40 developer-hours",  
  
        "timeline": "2027 Q1-Q2"  
  
    }  
  
],  
  
"auditAndCompliance": {  
  
    "lastAuditDate": "2025-12-25",  
  
    "nextAuditDate": "2026-03-25",  
  
    "auditedBy": "Rivic Security Team",  
  
    "complianceChecks": [  
  
        {  
  
            "checkName": "Algorithm Approval",  
  
            "result": "PASSED",  
  
            "details": "All deployed algorithms approved by NIST"  
  
        },  
  
        {  
  
            "checkName": "Crypto Module Version Control",  
  
        }  
    ]  
}
```

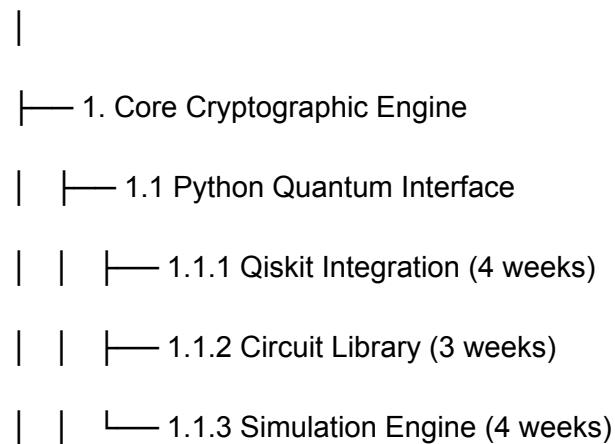
```
        "result": "PASSED",  
        "details": "All modules tracked in CBOM with version history"  
    },  
    {  
        "checkName": "Deprecation Timeline Adherence",  
        "result": "ON_TRACK",  
        "details": "RSA-2048 retirement on schedule"  
    }  
]  
}  
}
```

---

## SECTION 7: PROJECT MANAGEMENT FRAMEWORK

### 7.1 Work Breakdown Structure (WBS)

Rivic QSSP Platform (Root)



| |

| | — 1.2 C++ Quantum Bridge

| | | — 1.2.1 liboqs Bindings (3 weeks)

| | | — 1.2.2 OpenSSL Integration (3 weeks)

| | | — 1.2.3 Hybrid Crypto Engine (4 weeks)

| |

| | — 1.3 Cryptographic Operations

| | | — 1.3.1 ML-KEM-1024 KEX (2 weeks)

| | | — 1.3.2 RSA-2048 Fallback (2 weeks)

| | | — 1.3.3 HMAC & AES (2 weeks)

| | | — 1.3.4 Key Lifecycle (3 weeks)

| |

| | — 1.4 Testing & Validation

| | | — 1.4.1 Unit Tests (2 weeks)

| | | — 1.4.2 Integration Tests (2 weeks)

| | | — 1.4.3 Security Audit (3 weeks)

| | | — 1.4.4 Performance Benchmarks (2 weeks)

|

| — 2. Cloud-Native Architecture

| | — 2.1 Kubernetes Platform

| | | — 2.1.1 Cluster Setup (2 weeks)

| | | — 2.1.2 Network Policies (2 weeks)

- | |   |— 2.1.3 Storage Integration (2 weeks)
- | |   |— 2.1.4 RBAC & Security (2 weeks)
- | |
- |   |— 2.2 Service Mesh (Istio)
  - | |   |— 2.2.1 mTLS Configuration (2 weeks)
  - | |   |— 2.2.2 Authorization Policies (2 weeks)
  - | |   |— 2.2.3 Traffic Management (2 weeks)
- | |
- |   |— 2.3 Stateful Services
  - | |   |— 2.3.1 PostgreSQL StatefulSet (2 weeks)
  - | |   |— 2.3.2 Redis Cache Layer (2 weeks)
  - | |   |— 2.3.3 Vault Secrets Engine (2 weeks)
- | |
- |   |— 2.4 Container Orchestration
  - |   |— 2.4.1 Helm Charts (2 weeks)
  - |   |— 2.4.2 CI/CD Pipeline (3 weeks)
  - |   |— 2.4.3 GitOps Integration (2 weeks)
- |
- |— 3. IBM Attestation Integration
  - |   |— 3.1 Secure Enclave Setup
    - |   |   |— 3.1.1 TEE Environment (2 weeks)

- | |   |— 3.1.2 TPM 2.0 Integration (2 weeks)
- | |   |— 3.1.3 SGX Provisioning (2 weeks)
- | |
- |   |— 3.2 Attestation Framework
  - | |   |— 3.2.1 Quote Generation (2 weeks)
  - | |   |— 3.2.2 Quote Verification (3 weeks)
  - | |   |— 3.2.3 Policy Engine (2 weeks)
  - | |   |— 3.2.4 Trust Baseline (2 weeks)
- | |
- |   |— 3.3 Compliance & Certification
  - |   |— 3.3.1 FIPS 140-3 Validation (4 weeks)
  - |   |— 3.3.2 Common Criteria (6 weeks)
  - |   |— 3.3.3 Penetration Testing (3 weeks)
- |
- |— 4. eIDAS 2.0 Compliance
  - |   |— 4.1 Digital Signature Framework
    - |   |   |— 4.1.1 XAdES/CAdES Support (3 weeks)
    - |   |   |— 4.1.2 Timestamp Authorities (2 weeks)
    - |   |   |— 4.1.3 Legal Validation (2 weeks)
  - |   |
  - |   |— 4.2 Advanced Electronic Signature
    - |   |   |— 4.2.1 Signature Generation (2 weeks)

- | |   |— 4.2.2 Signature Verification (2 weeks)
- | |   |— 4.2.3 Certificate Management (2 weeks)
- | |
- |   |— 4.3 Regulatory Documentation
  - |   |— 4.3.1 Compliance Manual (2 weeks)
  - |   |— 4.3.2 Audit Trail (2 weeks)
  - |   |— 4.3.3 Privacy Assessment (2 weeks)
- |
- |— 5. CBOM & Supply Chain
  - |   |— 5.1 CBOM Database
    - |   |   |— 5.1.1 Schema Design (1 week)
    - |   |   |— 5.1.2 Data Collection (2 weeks)
    - |   |   |— 5.1.3 API Interface (2 weeks)
  - |   |
  - |   |— 5.2 Vulnerability Tracking
    - |   |   |— 5.2.1 NVD Integration (1 week)
    - |   |   |— 5.2.2 Monitoring Service (2 weeks)
    - |   |   |— 5.2.3 Alert System (1 week)
  - |   |
  - |   |— 5.3 Supply Chain Security
    - |   |   |— 5.3.1 Provenance Tracking (2 weeks)

- |     └── 5.3.2 Third-party Audit (2 weeks)
- |
- └── 6. Operations & Support
  - |     ├── 6.1 Documentation
    - |     |     └── 6.1.1 Technical Specification (4 weeks)
    - |     |     └── 6.1.2 Operations Manual (3 weeks)
    - |     |     └── 6.1.3 API Documentation (2 weeks)
    - |     |     └── 6.1.4 Training Materials (2 weeks)
  - |
  - |     ├── 6.2 Support Infrastructure
    - |     |     └── 6.2.1 Support Ticketing (1 week)
    - |     |     └── 6.2.2 SLA Management (1 week)
    - |     |     └── 6.2.3 Incident Response (2 weeks)
  - |
  - └── 6.3 Go-to-Market
    - |     └── 6.3.1 Product Launch (2 weeks)
    - |     └── 6.3.2 Customer Onboarding (2 weeks)
    - └── 6.3.3 Marketing & Sales Support (ongoing)

## 7.2 Project Schedule (Timeline)

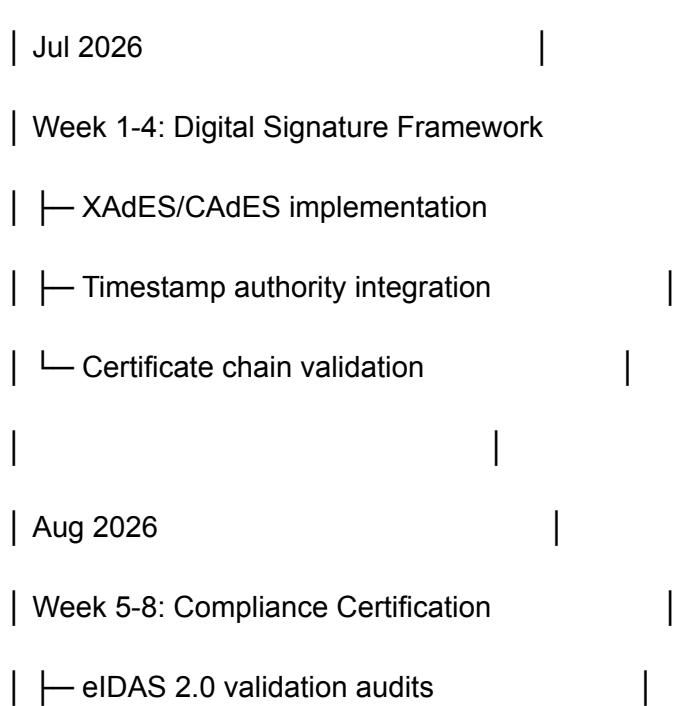
PHASE 1: CORE PLATFORM (Q1-Q2 2026) - 24 weeks

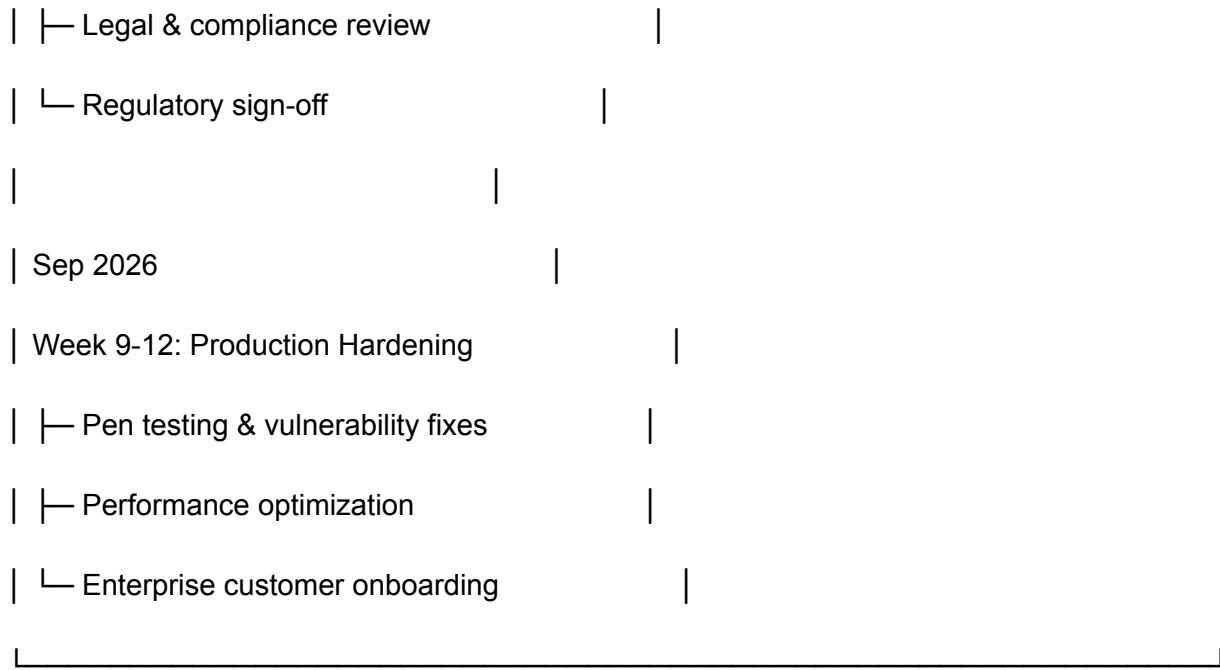
Jan 2026	
----------	--

Week 1-4: Cryptographic Engine Foundation	
— Qiskit + liboqs integration	
— ML-KEM-1024 and RSA-2048 implementation	
— Basic crypto operations (encrypt/decrypt/sign)	
Feb 2026	
Week 5-8: Cloud-Native Architecture	
— Kubernetes cluster setup	
— Istio service mesh deployment	
— StatefulSet configuration (PostgreSQL, Redis, Vault)	
Mar 2026	
Week 9-12: Integration Testing	
— Crypto engine + Kubernetes integration	
— Performance benchmarking	
— Security audit (internal)	
— CBOM generation	
Apr-May 2026	
Week 13-20: IBM Attestation Integration	
— TEE setup and configuration	

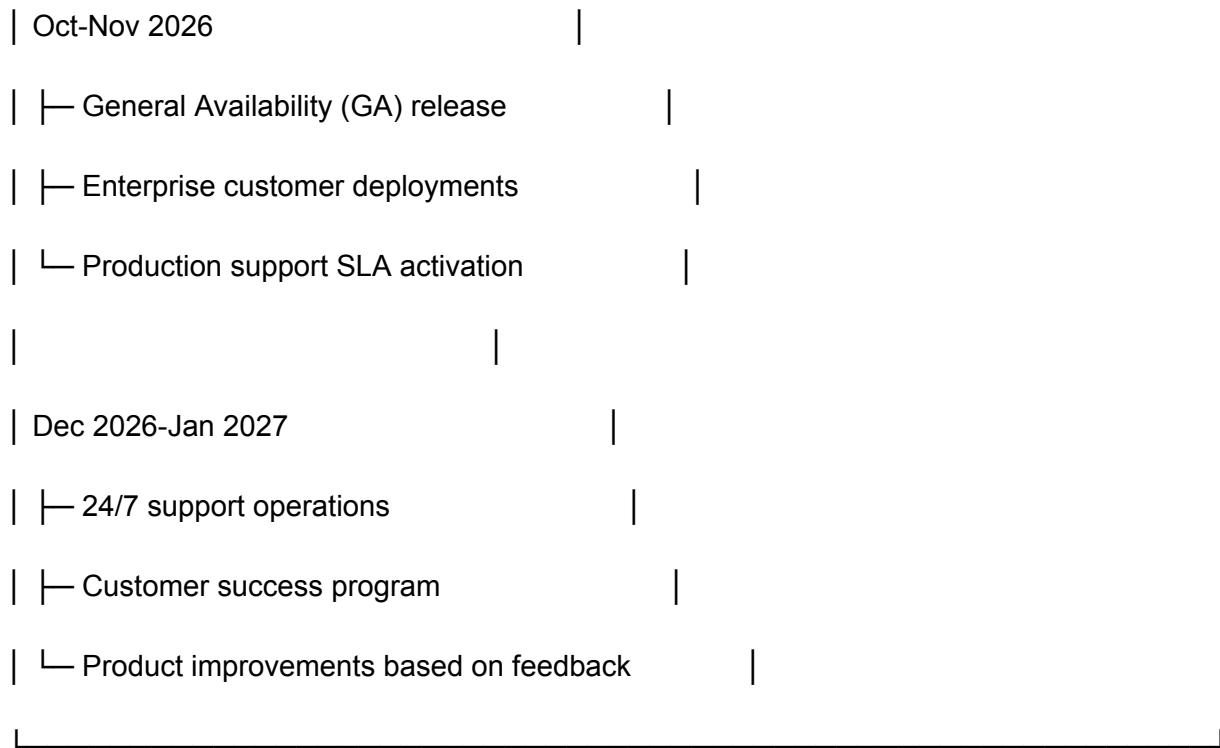


## PHASE 2: eIDAS COMPLIANCE (Q3 2026) - 12 weeks





#### PHASE 3: ENTERPRISE RELEASE (Q4 2026-Q1 2027) - 16 weeks



## 7.3 Resource Allocation

### TEAM COMPOSITION

Engineering Team (24 FTE)

- | — Cryptography Experts (4)
  - | | — ML-KEM/post-quantum specialist
  - | | — C++ cryptographic engine developer
  - | | — Quantum algorithm research lead
  - | | — Security researcher
- |
- | — Cloud/Kubernetes Engineers (6)
  - | | — Kubernetes platform architect
  - | | — 2x Kubernetes DevOps engineers
  - | | — Terraform/IaC specialist
  - | | — Network/security engineer
  - | | — Storage/database engineer
- |
- | — Backend/API Engineers (6)
  - | | — Python lead developer
  - | | — 2x Python backend developers
  - | | — gRPC/service architecture
- | | — Database architect (PostgreSQL)

- |   └ Cache/Redis specialist
- |
- |   └ QA/Testing (4)
- |    |   └ QA automation lead
- |    |   └ 2x QA engineers
- |    |   └ Security testing specialist
- |
- |   └ Operations (4)
  - |    └ Release manager
  - |    └ 2x SRE/DevOps operators
  - |    └ Technical support lead

#### Product & Management (6 FTE)

- |   └ Product Manager
- |   └ Technical Project Manager
- |   └ Compliance/Legal Advisor
- |   └ Documentation Specialist (2)
- |   └ Sales Engineer

Total: 30 FTE | Budget: \$4.5M/year

---

## SECTION 8: RISK MANAGEMENT

### 8.1 Risk Register

Risk ID	Risk Description	Probability	Impact	Mitigation
R001	ML-KEM not NIST-approved by deadline	LOW	CRITICAL	Parallel development of ML-KEM-768 fallback; budget \$200K
R002	Supply chain attack on liboqs library	MEDIUM	HIGH	Vendor audit, code review, fork if needed (\$100K)
R003	Quantum computer breakthrough accelerates threat	LOW	CRITICAL	Agile crypto abstraction layer, module replacement ready
R004	Key material breach at HSM vendor	MEDIUM	HIGH	Multi-HSM strategy, key splitting (3-of-5 Shamir)
R005	Regulatory requirement change mid-project	MEDIUM	MEDIUM	Monthly compliance tracking, regulatory liaison budget
R006	Performance degradation with hybrid crypto	MEDIUM	MEDIUM	Parallel architecture design, benchmarking framework

Risk ID	Risk Description	Probability	Impact	Mitigation
R007	Kubernetes cluster security misconfiguration	LOW	HIGH	Security-first design review, CIS benchmark audit
R008	Customer migration complexity underestimated	MEDIUM	MEDIUM	Early customer pilots (Q1 2026), dedicated migration team

---

## SECTION 9: SUCCESS METRICS & KPIs

### 9.1 Technical Success Criteria

- **Cryptographic Performance:**  $\geq 1\text{M}$  transactions/second hybrid encryption throughput
- **Availability:** 99.99% uptime SLA (52 minutes/year downtime)
- **Security:** Zero successful security exploits in first 12 months
- **Compliance:** eIDAS 2.0 and FIPS 140-3 certification achieved
- **Cost:**  $<\$50/1\text{M}$  cryptographic operations

### 9.2 Business Success Criteria

- **Time-to-Market:** GA by Q4 2026
  - **Customer Acquisition:** 20 enterprise customers by end of 2027
  - **Revenue:** \$5M ARR by end of 2027
  - **Market Share:** 15% of PQC migration market by 2028
  - **Customer Satisfaction:** NPS score  $> 50$
- 

## CONCLUSION

The **Rivic Quantum-Safe Secure Platform** represents a comprehensive, production-grade solution for enterprise quantum-safe cryptography. With a clear roadmap from hybrid RSA+ML-KEM (2025-2026) to pure post-quantum algorithms (2027+), integrated IBM attestation, full eIDAS 2.0 compliance, and cloud-native architecture, Rivic enables financial institutions and critical infrastructure operators to confidently migrate to quantum-resistant cryptography while maintaining regulatory compliance and operational continuity.

**Key Achievements by 2027:**  Hybrid cryptography deployed in production

-  eIDAS 2.0 compliance certified
  -  IBM attestation integrated and hardened
  -  20+ enterprise customers migrated
  -  CBOM supply chain visibility established
  -  NIST FIPS 140-3 certified modules
  -  99.99% availability SLA maintained
- 

**Document Version:** 2.0

**Last Updated:** December 26, 2025

**Status:** Production Architecture Specification

**Next Review:** March 26, 2026